Complexity analysis for Prediction Learning model Code

Looking at our machine learning model code, and focusing on it its time and space complexity, we decided to cut the code into a few parts and look at each alone then we collect what we have seen from all of that to make a general conclusion about the complexity of our code.

Starting with the “import” we have at the start of our code, which usually takes a linear time and space complexity depending on how big is the imported library in its size.

Snapshot from the code:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import pandas_datareader as web
import datetime as dt
from tkinter import *
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM
```

The second major library we have used is the “DateTime” Library, and after checking the whole code of the library, we found out that it mainly and mostly depends on two main things which are single for loops and if statements which also have linear O(n) time complexity. In addition to that, the space complexity of the If statements and the single for loops are also O(n), and there is a reason for that.

Snapshot from the code:

```
start = dt.datetime(2013, 1, 1)
    end = dt.datetime(2020,1,1)
```

In order to measure the space complexity, we may need to look at how much extra space is required in order to hold all the information necessary to run and execute the code. In our case of if statements and for loops, the code needs extra space to store

1-The values of temporary calculations in the loop,

2-The value of the loop variable,

3- The other miscellaneous like program counter, and so on.

Notice that all of the three have O(n) space complexity, even the first one which we can reuse its space over each loop.

After that, we have the pandas_datareader module, which is used to read data from the yahoo finance website after scrapping it, and in order to answer how long does it takes to get the data from a website we should answer the following questions first:

1- How many pages you are getting?

2- What is the website URL you are crawling? How fast is that?

3- Are you using Agenty proxies or your own?

4- Do you have JavaScript / Images load enabled in your web scraping engine?

And depending on that we shall know the time and sometimes the space complexity of that module which is rational.

Snapshot from the code:

```
    test_data = web.DataReader(company, 'yahoo', test_start, test_end)
```

Usually, scrapping a webpage/URL that has no challenges to access (Normal case) may take a few seconds depending on how fast is your internet connection, and this is our case.

Then we have the numpy module, which after reviewing its source code and data structures, we found that it has a mix of time complexities between O(1) and O(n). In addition to that, the module doesn't have any nested for or while loop which is a good thing that makes us sure that it doesn't have any nested complexities like n^2.

Maybe we have been analyzing the code in a general view, but we still have to look at its data structure, the smaller and the main building units which may give us a different analysis. Look at the following table:

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)

In our code, we used array as one of the main lists data structures, and as we see in general that the worst-case scenario in the array row and in the list, in general, is O(n) for time complexity and also O(n) also for space complexity. We do have some single for loops in our code that has an O(n) complexity and also it appends to some types of lists/arrays in each loop, which also have an O(n) space complexity and O(n) time complexity if it is an array.

Last but not least, our model training code, and inside that model, the application is using the TensorFlow module, and one of the module functions we are using which is the “fit” function, looks like the following:

```
# coding=utf-8
def outer_factory():
    self = None
    step_function = None

    def inner_factory(ag__):

        def tf__train_function(iterator):
            """Runs a training execution with a single step."""
            with ag__.FunctionScope('train_function', 'fscope', ag__.ConversionOptio
                do_return = False
                retval_ = ag__.UndefinedReturnValue()
                try:
                    do_return = True
                    retval_ = ag__.converted_call(ag__.ld(step_function), (ag__.ld(s
                except:
                    do_return = False
                    raise
                return fscope.ret(retval_, do_return)
        return tf__train_function
    return inner_factory
```

This method is called when we call the line of code (`model.fit(x_train, y_train, epochs=25, batch_size=32)`), and it has two recursion calls, each recursion call has a time complexity of O(n^2), and being in a nested function and so on nested recursion calls, we can say that our time complexity for this training model is approaching O(n^4) in its worst of worst cases in our code.

Taking a deep look at the code above, we will realize that the code is reusing its variables in each recursive call, which means that the space complexity of this can be O(1), and O(n) in the worst of worst cases if this code is using any other methods that have data structures like the ones we have mentioned in the above table.

Conclusion:

From the above analysis, Our code mostly has a time complexity of O(n^4) and a space complexity of O(n).